Složitější datové typy

Složitější datové typy

- OBJEKT zelva.forward(100)
- STRING (textový řetězec) "abcd"

  neměnitelná
- TUPLE (n-tice) ("NPRG031", 1)

  neměnitelná
- LIST (seznam) [1, 2, 5, 3, 2]

  seznam s pořadím
- SET (množina) { 1, 5, 2 }

  bez pořadí

  každá hodnota je nebo není (max jednou)
- DICTIONARY (slovník) { 1:"a", 5:"xyz", 2:22 }

  dvojice klíč:hodnota

Objekt (první pohled)

Obsahuje DATA a FUNKCE

Přístup k nim pomocí tečky

```
>>> a = 128
>>> a.bit_length()
8
>>> [1,2,1,3,1,4].count(1)
3
```

Skoro všechno v Pythonu je objekt.

Podrobnosti někdy jindy.

Textový řetězec (první pohled)

```
>>> "ahoj"
'ahoj'
>>> 'ahoj'
'ahoj'
>>> type('ahoj')
<class 'str'>
```

Uvozovky nebo apostrofy... (nebo trojice téhož).

Textový řetězec

Jednotlivé znaky jsou v Unicode.

Pokud má obsahovat ohraničující nebo zvláštní znaky, tak zpětná lomítka.

Při dosazování se nekopíruje! (Dosazuje se odkaz.)

Operátor „is" (pro všechny objekty, nejen string):

```
>>> a = "abcd"
>>> b = a
>>> b is a # odkazuje na tentýž objekt?
True
```

Operátor „is“ (pro všechny objekty, nejen string)

```
>>> a = "abcd"
>>> b = "abcd"
>>> b is a # odkazuje na tentýž objekt?
True
```

Ale

```
>>> a = a + "e"
>>> b = b + "e"
>>> b == a # je to stejná hodnota?
True
>>> b is a # odkazuje na tentýž objekt?
False
```

Textový řetězec

Je neměnitelný (immutable),
pokud chceme část změnit, tak vyrobit nový řetězec.

Přístup k jednotlivým znakům pomocí indexu (pořadového čísla):

```
>>> a = "abcd"
>>> a[0]
'a'
>>> a[1]
'b'
```

Výřez (slice)

řetězec[*odkud*:*pokud*]

Indexy jsou od nuly

pozice pokud už se nezahrne:

```
>>> a = "abcd"
>>> a[1:3]
'bc'
```

Výřez (slice) – indexy

indexy počítané od konce:

```
>>> a = "abcd"
>>> a[-1]
'd'

>>> a = "abcd"
>>> a[1:-1]
'bc'

>>> "ABCDEFGH"[-3:-1]
'FG'
```

Jeden znak...

Python NEMÁ typ pro znak.

Znak je řetězec dlouhý jeden znak.

```
>>> a = "abcd"
>>> a[0:1]
'a'

>>> a[0]
'a'
>>> a[-1]
'd'
```

Vynechané parametry

od začátku:

```
>>> a[:3]
'abc'
```

až do konce:

```
>>> a[2:]
'cd'
```

```
>>> a[-3:]
'bcd'
```

Třetí parametr - krok

krok:

```
>>> a = "abcdefghijkl"
>>> a[::2]
'acegik'
>>> a[3::2]
'dfhjl'
```

Třetí parametr - krok

co bude výsledkem?

```
a[::-1]
```

Kód znaku

každému znaku, který patří do znakové sady, je přiřazeno určité číslo, tzv. Ordinální číslo znaku

standardní funkce:

```
ord( c ) znak -> int
chr( x ) int -> znak
```

Kód znaku

```
>>> ord( "A" )
65
>>> chr( 33 )
'!'
>>> ord( "圓" )
22291
>>> chr( 22291 )
'圓'
>>> chr( -5 )
ValueError: chr() arg not in
range(0x110000)
```

Porovnávání řetězců

Rovnost/nerovnost je snadná.

Větší/menší:

Porovnává se lexikograficky

Kódy znaků jsou podle Unicode, platí:

'0','1',...'9'

v tomto pořadí a bezprostředně za sebou

'A','B',...,'Z'

v tomto pořadí

Porovnávání podle národních pravidel je složité!

cis	cíl
cíl	cis
čáp	čáp
definice	ďábel
děda	decibel
délka	děda
dělitel	definice
ďábel	dělitel
decibel	délka
demokracie	demokracie
démon	démon
demontáž	demontáž

```
>>> import locale
>>> locale.setlocale(locale.LC_ALL,
'cs_CZ.utf8')
>>> locale.strxfrm('sloda') <
locale.strxfrm('schody')
```

Ještě řetězce…

Je to trochu složitější:

- str je posloupnost znaků
- každý znak je zapsán několika bajty (různé počty)
- vztah znak-bajty určuje použité kódování
- typ bytes

https://betterprogramming.pub/strings-unicode-and-bytes-in-python-3-everything-you-always-wanted-to-know-27dc02ff2686

Otázka:

jak zapsat podmínku

„znak Z je číslice"

?

Odpověď 1:

```
(Z='0') or (Z='1') or...or (Z='9')
```

Odpověď 2:

```
(Z>='0') and (Z<='9')
```

Hodnota čísla uloženého ve stringu

LIST (seznam)

Seznam hodnot, zachovává pořadí, indexy od 0.
Vytvoření: v [ ], oddělený čárkami.
Přidávání: .append(co), .insert(kam, co), .extend(seznam)
Může současně obsahovat hodnoty různých typů.
Záporné indexy a slice stejné jako u typu string.

```
>>> seznam = [ 10, 'xyz', 30 ]
>>> seznam[2]
30
>>> seznam.append(45)
>>> seznam
[10, 'xyz', 30, 45]
```

LIST (seznam) 2.

...a problémy ! (seznam vs. odkaz na seznam):

```
>>> A = [1,2,3,4]
>>> A
[1, 2, 3, 4]
>>> B = A          # TADY SE NEKOPIRUJE!
>>> B[0] = 29      # "DOSAZUJU DO B...
>>> A
[29, 2, 3, 4]
```

POZOR: Proměnná neobsahuje hodnotu !

Je to jen jméno (odkaz) na hodnotu !

LIST (seznam) 3.

JAK VYTVOŘIT KOPII SEZNAMU:

```
>>> A = [1,2,3,4]
>>> A
[1, 2, 3, 4]
>>> B = A[:]   # NOVÝ SEZNAM = KOPIE
>>> B[0] = 29 # "DOSAZUJU DO B...
>>> A
[1, 2, 3, 4]
```

nebo jinak:

```
>>> C = LIST(A) # NOVÝ SEZNAM = KOPIE
```

LIST (seznam) 4.

(neboli Pythonovské triky, je jich spousta)

Vytvoření pomocí násobení (jde i u typu string).

```
>>> a = 10*[0]
>>> a
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>> b = 5*[1, 2]
>>> b
[1, 2, 1, 2, 1, 2, 1, 2, 1, 2]
```

LIST (seznam) 5.

ALE zase pozor na odkaz versus hodnota:

```
>>> a = 2*[[7,8,9]]
>>> a
[[7, 8, 9], [7, 8, 9]]


>>> a[0][1] = 'x'
>>> a
[[7, 'x', 9], [7, 'x', 9]]
```

LIST (seznam) 6.

Hledání v seznamu:

```
>>> 3 in [1, 2, 3]
True

>>> [1, 2, 3].index(2)
1
>>> [1, 2, 3].index(77)
ValueError: 77 is not in list
```

LIST (seznam) 7.

Mazání ze seznamu:

```
>>> a = [10,20,30,40]
>>> a.remove(20)
>>> a
[10, 30, 40]
>>> del a[:2]
[40]
```

Cyklus FOR

Když známe dopředu hodnoty, které chceme procházet.

```
for proměnná in seznam:
    příkaz
    příkaz
```

```
>>> for x in [1, 2, "A", [10,20]]:
        print( x )
1
2
A
[10, 20]
```

Funkce range

…jako by vracela LIST
(v Pythonu 2 ještě tak bylo, ale Python 3 to dělá efektivněji).

range(první, první_který_už_ne)

```
>>> for x in range(1,5):
        print( x )
1
2
3
4
```

Funkce range

První parametr se dá vynechat, potom je od 0:

```
>>> for x in range(5):
        print( x )
0
1
2
3
4
```

Funkce range

Taky můžeme přidat třetí parametr – krok:

```
>>> for x in range(0,10,2):
        print( x )
0
2
4
6
8
```

Příklady